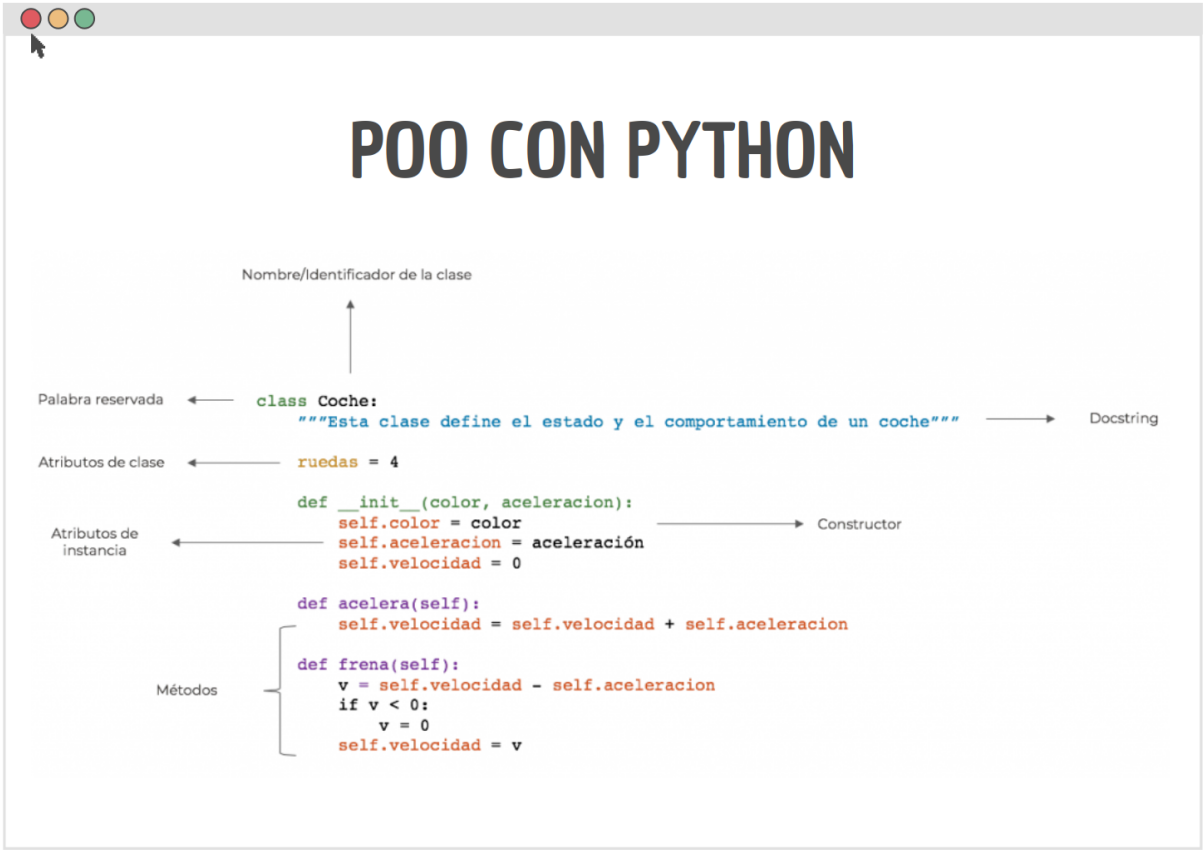
PROGRAMACIÓN WEB 2



Profesor(a):
Quispe Cruz, Marcela

Estudiantes:
Arias Quispe, Jhonatan David
Mamani Huarsaya, Jorge Luis
Velarde Saldaña Jhossep Fabritzio

Repositorio GitHub:
https://github.com/jorghee/OOP-with-python

28 de mayo, 2024

El programa Agenda telefónica

- El programa necesita guardar los datos que el usuario dispone, estos datos deben ser almacenados en algún tipo de archivos. Debido a que se necesitan guardar listas de objetos, se ha decidido hacer uso de archivos JSON, un formato ligero de intercambio de datos, fácil de leer y escribir con la biblioteca json de Python, adecuado para datos estructurado.
- Leer y escribir en archivos con Python viene de forma nativa y se utiliza a través de las funciones integradas `open`, `read`, `write` entre otras. Sin embargo para manejar ciertos formatos especificos como JSON, Python ofrece el modulo estándar json.

La clase Contacto

Contiene los campos mencionados necesarios para identificar al contacto

```
def __init__(self, nombre: str, telefono: str, direccion: str, relacion: str):
  self._nombre = nombre
  self._telefono = telefono
  self._direccion = direccion
  self._relacion = relacion
```

Además debe de contener los métodos que se llaman automaticamente al momento de representar como string al objeto actual.

```
def __repr__(self) -> str:
  return self.__str__()

def __str__(self) -> str:
  return (f"\nNombre:\t\t{self._nombre}\nTelefono:\t{self._telefono}\n"
          f"Direccion:\t{self._direccion}\nRelacion:\t{self._relacion}\n")
```

Debemos de tener un método que nos permita parsear el objeto actual como un diccionario. Por ejemplo, que el identificador de referencia al campo name sea como clave y su valor sea el correspondiente valor del campo.

```
def parsear_a_json(self):
  return {
    "nombre": self.nombre,
    "telefono": self.telefono,
    "direccion": self.direccion,
    "relacion": self.relacion
  }
```

Un método que reciba como argumento un diccionario, basicamente es el objeto JSON parseado por el modulo json de Python. Este diccionario contiene los campos de una instancia de la clase Contact, por lo tanto con estos datos se debe crear el nuevo objeto. Si analizamos, este método será parte de cargar los datos guardados en un archivo JSON.

```
@staticmethod
def parsear_de_json(dic):
  return Contacto(dic["nombre"], dic["telefono"], dic["direccion"], dic["relacion"])
```

El método debe de ser estático porque necesitamos acceder a este mismo sin tener que crear aún un objeto Contacto.

La clase Agenda

Esta clase contiene un campo `self._contacts` que es una lista de objetos Contacto. Aqui tenemos una particularidad a destacar. Se decidió que el **archivo que guarda los contactos se recupera automáticamente** al momento de crear una instancia de esta clase.

```
def __init__(self, path_data):
  self._contacts = self.recuperar_agenda(path_data)
```

Por lo tanto, comenzaremos explicando el método `recuperar_agenda()`.

El método `recuperar_agenda(path_data)`

La lógica es recuperar el archivo con la dirección que nosotros como desarrolladores han decidido, en caso no exista el archivo, entonces se regresa una lista vacíá; de lo contrario necesitamos cargar el archivo.

```
def recuperar_agenda(self, path_data):
  if not (os.path.exists(path_data)):
    return []

  with open(path_data, "r") as data:
    datos = json.load(data)
    print("Recuperación exitosa...\n")
  return [Contacto.parsear_de_json(dic) for dic in datos]
```

Cuando cargamos el archivo, este se carga en diccionarios, ya que la conversión de objetos JSON es a diccionarios en Python. Entonces aqui debemos de utilizar el método `parsear_de_json()` que recibe el diccionario y crear con estos valores el objeto Contacto.

El método `getContact(pattern)`

Básicamente la lógica consiste en iterar sobre las lista `self._contacts` y recuperar su campo `nombre`.

Ahora necesitamos comparar dicho nombre con el patrón ingresado. Para resolver este problema se ha utilizado el siguiente algoritmo.

Algoritmo de Knuth-Morris-Pratt (KMP): El algoritmo KMP utiliza la información del propio patrón para evitar retrocesos innecesarios en la cadena de texto. Esto se logra mediante la construcción de un arreglo de prefijo (función de prefijo o función π), también conocido como "arreglo de fallos" o "tabla de fallos", que se utiliza para saltar comparaciones redundantes.

```
KMP-MATCHER(T, P, n, m)
1  π = COMPUTE-PREFIX-FUNCTION(P, m)
2  q = 0                              // number of characters matched
3  for i = 1 to n                     // scan the text from left to right
4      while q > 0 and P[q + 1] ≠ T[i]
5          q = π[q]                   // next character does not match
6      if P[q + 1] == T[i]
7          q = q + 1                  // next character matches
8      if q == m                      // is all of P matched?
9          print "Pattern occurs with shift" i − m
10         q = π[q]                   // look for the next match

COMPUTE-PREFIX-FUNCTION(P, m)
1  let π[1 : m] be a new array
2  π[1] = 0
3  k = 0
4  for q = 2 to m
5      while k > 0 and P[k + 1] ≠ P[q]
6          k = π[k]
7      if P[k + 1] == P[q]
8          k = k + 1
9      π[q] = k
10 return π
```

Figure 1: Pseudocodigo del algoritmo KMP

Entonces en ves de devolver el indice de la primera posición de la cadena de texto de todas las coincidencias, nosotros hemos hecho que devuelva `True` o `False`.

```python
# Algoritmo Knuth-Morris-Pratt (KMP)
def _match(self, name, pattern):
  lps = self._compute_lps(pattern)
  i = 0
  j = 0

  while i < len(name):
    if pattern[j] == name[i]:
      i += 1
      j += 1

    if j == len(pattern):
      j = lps[j - 1]
      return True
    elif i < len(name) and pattern[j] != name[i]:
      if j != 0:
        j = lps[j - 1]
      else:
        i += 1
  return False

```

```
# Generación de la funcion £\pi£
def _compute_lps(self, pattern):
  lps = [0] * len(pattern)
  length = 0
  i = 1

  while i < len(pattern):
    if pattern[i] == pattern[length]:
      length += 1
      lps[i] = length
      i += 1
    else:
      if length != 0:
        length = lps[length - 1]
      else:
        lps[i] = 0
        i += 1
  return lps
```

El método `guardar_agenda()`

El método consiste en iterar en los objetos de la lists `self._contacts` y generar una lista ya no de objetos Contacto, sino ahora una lista de diccionarios, usando el método `parsear_a_json()` de la clase Contacto.

```
def guardar_agenda(self):
  contacts_pars = [contact.parsear_a_json() for contact in self._contacts]
  with open("./data/data.json", "w", encoding = "utf-8") as save:
    json.dump(contacts_pars, save, ensure_ascii=False, indent=2)
    print("Guardacion exitosa...\n")
```